System design: Quora

Suresh Podeti

11 min read · Oct 26, 2023

Photo by Rubaitul Azad on Unsplash

Introduction

With so much information available online, finding answers to questions can be daunting. That's why **information sharing** online has become so widespread..

Information-seeking through other people can be more instructive, even if it comes at the cost of additional time. Seeking information through search engines can lead

to dead ends because of content unavailability on a topic. Instead, we can ask questions of others.

Quora is a **social question-and-answer service** that allows users to ask questions to other users. Quora was created because of the issue that asking questions from search engines results in fast answers but shallow information. Instead, we can ask the general public, which feels more conversational and can result in deeper understanding, even if it's slower. Quora enables anyone to ask questions, and anyone can reply. Furthermore, there are domain experts that have in-depth knowledge of a specific topic who occasionally share their expertise by answering questions.

Requirements

Functional

- **Questions and answers**: Users can ask questions and give answers. Questions and answers can include images and videos.
- **Upvote/downvote and comment**: It is possible for users to upvote, downvote, and comment on answers.
- **Search**: Users should have a search feature to find questions already asked on the platform by other users.
- **Recommendation system**: A user can view their feed, which includes topics they're interested in. The feed can also include questions that need answers or answers that interest the reader.
- **Ranking answers**: We enhance user experience by ranking answers according to their usefulness. The most helpful answer will be ranked highest and listed at the top.

Non-Functional

- **Scalability**: The system should scale well as the number of features and users grow with time. It means that the performance and usability should not be impacted by an increasing number of users.
- **Consistency**: The design should ensure that different users' views of the same content should be consistent. In particular, critical content like **questions and answers should be the same for any collection of viewers**. However, it is not

necessary that all users of Quora see a newly posted question, answer, or comment right away.

- **Availability:** The system should have high availability. This applies to cases where servers receive a large number of concurrent requests.
- **Performance:** The system should provide a smooth experience to the user without a noticeable delay.

Components

Web and application servers

A typical **Quora page is generated by various services.** The web and application servers maintain various processes to generate a webpage.**Web servers generate part of the web page** and let the **worker process in the application servers do the rest of the page** generation.

The web servers have **manager processes** and the application servers have **worker processes** for handling various requests. The manager processes distribute work among the worker processes using a router library. The router library is enqueued with tasks by the manager processes and dequeued by worker processes. Each application server maintains several in-memory queues to handle different user requests.

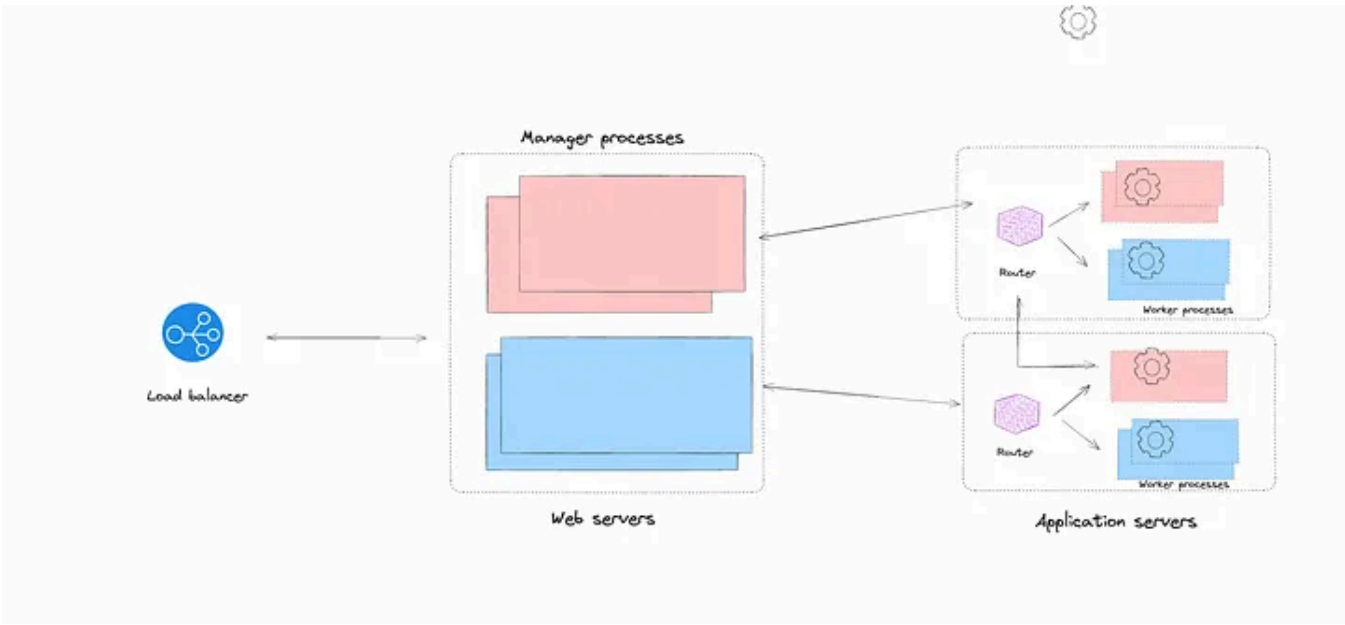


Fig 1.0: Web server and application servers at Quora

Data stores

- **Relational** — We can use critical data like **questions, answers, comments, and upvotes/downvotes** in a relational database like **MySQL** because it offers a higher degree of consistency.

- **NoSQL —** NoSQL databases like **HBase** can be used to store the **number of views of a page, scores** used to rank answers, and the extracted features from data to be used for recommendations later on. Because recomputing features is an expensive operation, HBase can be a good option to store and retrieve data at high bandwidth. We require high read/write throughput because big data processing systems use high parallelism to efficiently get the required statistics.

- **Blob storage** — to store **videos and images** posted in questions and answers.

Distributed cache

- **Memcached —** to store frequently accessed critical data that is otherwise stored in MySQL

- **Redis —** mainly used to store an online view counter of answers because it allows in-store increments

- **CDNs —** serve frequently accessed videos and images

Compute servers

A set of compute servers are required to facilitate features like recommendations and ranking based on a set of attributes. These features can be computed in online or offline mode. The compute servers use machine learning (ML) technology to provide effective recommendations. Naturally, these compute servers have a substantially high amount of RAM and processing power.

Of course, other basic building blocks like load balancers, monitoring services, and rate limiters will also be part of the design.

High level design

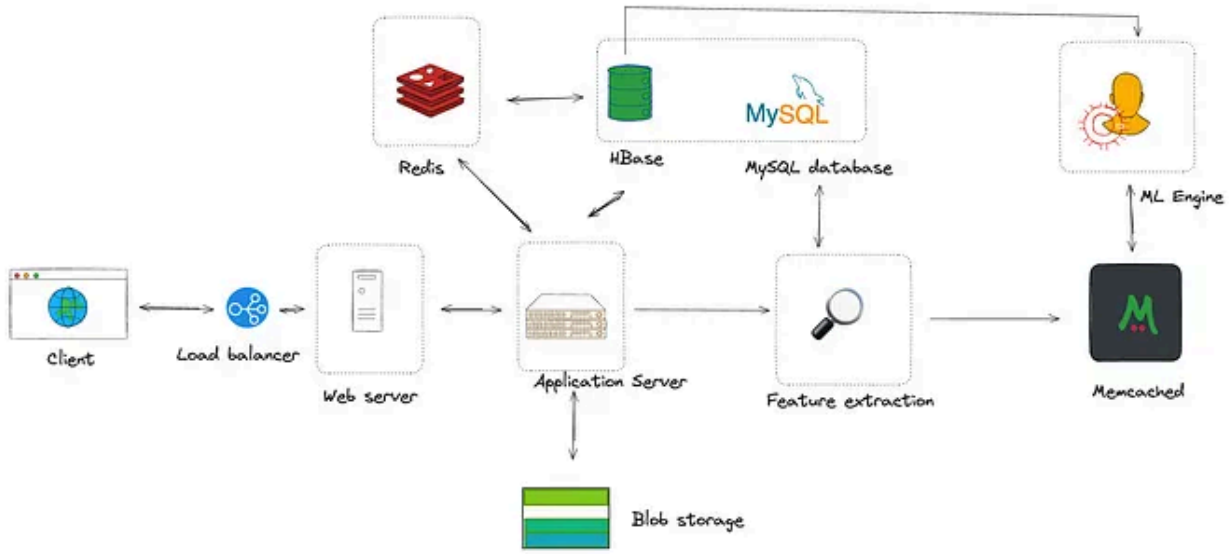


Fig 2.0: High level design of Quora

Posting question, answers, comments

The web servers receive user requests through the load balancer and direct them to the application servers. Meanwhile, the **web servers generate part of the web page** and let the **worker process in the application servers do the rest of the page** generation. The **questions and answers** data is stored in a **MySQL** database, whereas any **videos and images** are stored in the **blob storage**. A similar approach is used to post comments and upvote or downvote answers.

Task **prioritization** is performed by employing different queues for different tasks. We perform prioritization because certain tasks require immediate attention — for example, fetching data from the database for a user request — while others are not so urgent — for example, sending a weekly email digest. The worker processes will perform tasks by fetching from these queues.

Answer ranking system

Answers to questions can be sorted based on date. Although it is convenient to develop a ranking system on the basis of date (using time stamps), users prefer to see the **most appropriate answer at the top**. Therefore, Quora uses **ML to rank answers**. Different features are extracted over time and stored in the **HBase** for each type of question. These features are forwarded to the ML engine to rank the most useful answer at the top. We cannot use the number of **upvotes** as the only metric for ranking answers because a **good number of answers can be jokes** — and such answers also get a **lot of upvotes**. It is good to implement the ranking system offline because good answers get upvotes and views over time. Also, the offline mode poses a lesser burden on the infrastructure.

Implementing the ranking system offline and the need for special ML hardware makes it suitable to use some public cloud elastic services.

Recommendation system

The recommendation system is responsible for several features. For example, we might need to develop a user feed, find related questions and ads, recommend questions to potential respondents, and even highlight duplicate content and content in violation of the service's terms of use. Unlike the answer ranking system, the recommendation system must provide **both online and offline** services. This system receives requests from the application server and forwards selected features to the ML engine.

Search feature

Over time, as questions and answers are fed to the Quora system, it is possible to build an index in the HBase. User search queries are matched against the index, and related content is suggested to the user. Frequently accessed indexes can be served from cache for low latency. The index can be constructed from questions, answers, topics labels, and usernames.

Limitations

Limitations of web and application servers

To entertain the user's request, payloads are transferred between web and application servers, which increases latency because of **network I/O between these two types of server**s. Even if we achieve parallel computation by separating the web from application servers (that is, the manager and worker processes), the added latency due to an additional network link erodes a user's experience. Apart from data transfer, control communication between the router library with manager and worker processes also imposes additional performance penalties

In-memory queue failure

The internal architecture of **application servers** log tasks and **forward them to the in-memory queues**, which serve them to the workers. These in-memory queues of different priorities can be **subject to failures**. For instance, if a queue gets lost, all the tasks in that queue are lost as well, and manual engineering is required to recover those tasks. This greatly reduces the performance of the system. On the other hand, replicating these queues requires increasing RAM size. Also, with the number of features (functional requirements) that our system offers, many tasks can get assembled, which results in insufficient memory. At the same time, it is **not desirable to choke application servers with not-so-urgent tasks.**

Increasing QPS on MySQL

Because we have a higher number of features offered by our system, few MySQL tables **receive a lot of user queries.** This results in a higher number of QPS on certain MySQL servers, which can result in higher latency.

Latency of HBase

Even though HBase allows high real-time throughput, its P99 latency is not among the best. A number of Quora features require the ML engine that has a latency of its own. Due to the addition of the higher latency of HBase, the overall performance of the system degrades over time.

Solutions

Service hosts

We combine the **web and application servers** within a **single powerful machine** that can handle all the processes at once. This technique eliminates the network I/O and the latency introduced due to the network hops required between the manager, worker, and routing library processes.

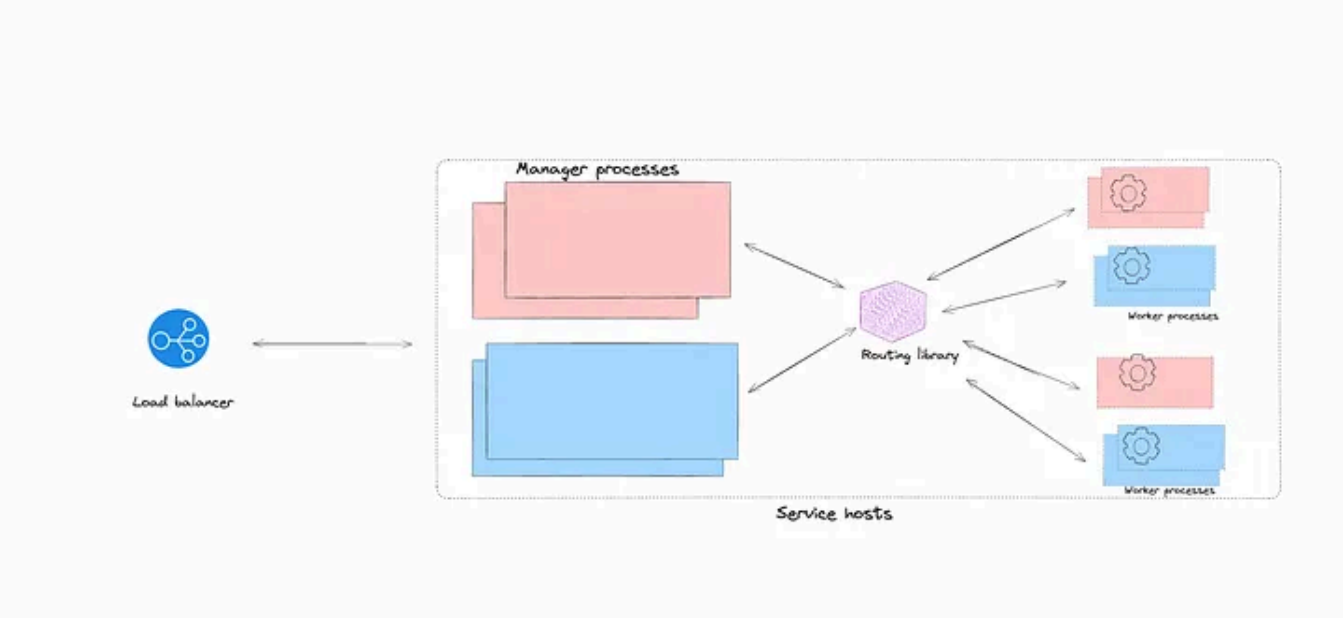


Fig 3.0: The updated design, where web and application servers are combined in the service host

Vertical sharding of MySQL

Tables in the MySQL server are converted to separate shards that we refer to as **partitions.** A partition has a **single primary server** and **multiple replica servers.**

The **goal** is to improve performance and **reduce the load due to an increasing number of queries on a single database table.** To achieve that, we do vertical sharding in two ways:

1. We split tables of a single database into multiple partitions. The concept is depicted in Partitions 2 and 3, which embed Tables 4 and 3, respectively.

2. We combine multiple tables into a single partition, where `join` operations are anticipated. The concept is depicted in Partition 1, which embeds Tables 1 and 2.

Therefore, we are able to co-locate related data and reduce traffic on hot data. The illustration below depicts vertical sharding at Quora.

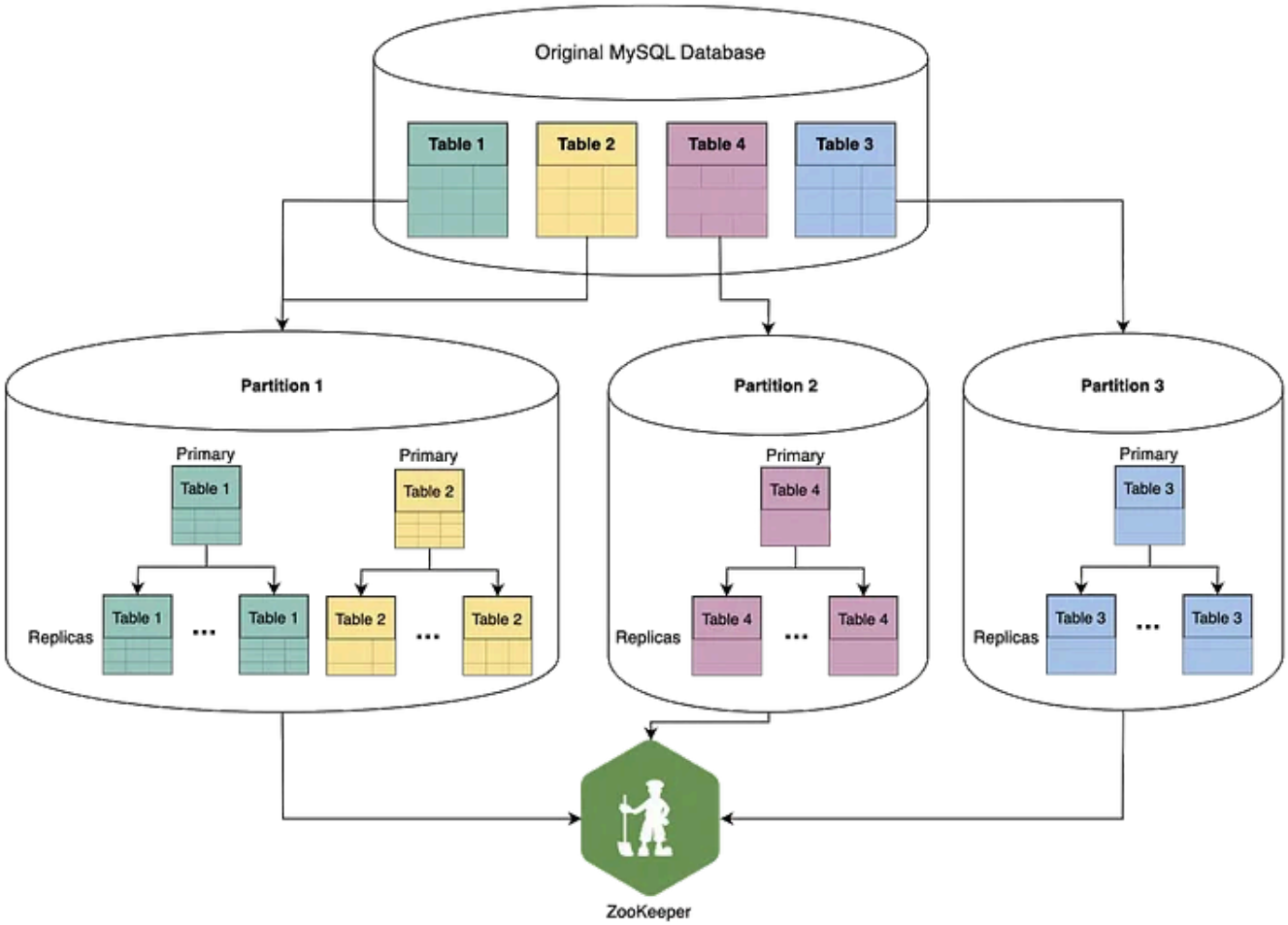


Fig 4.0: The architecture of vertical sharding at Quora

> ***Note:*** *Vertical sharding is of particular interest in Quora's design because horizontal sharding is more common in the database community. The main idea behind vertical sharding is to achieve scalability by carefully dividing or re-locating tables and eliminating join operations across different shards. Nevertheless, a vertically sharded partition or table can grow horizontally to an extent where horizontal sharding will be necessary to retain acceptable performance.*

MyRocks

The new design embeds MyRocks as the key-value store instead of HBase. We use the MyRocks version of RocksDB for two main reasons:

1. MyRocks has a lower p99 latency instead of HBase. Quora claims to have reduced P99 latency from 80 ms to 4 ms using MyRocks.

2. There are operational tools that can transfer data between MyRocks and MySQL.

> **Note:** Quora serves the ML compute engine by extracting features from questions and answers stored in MySQL. In this case, the operational tools come in handy to transfer data between MyRocks and MySQL.

Kafka

Our updated design reduces the request load on service hosts by separating not-so-urgent tasks from the regular API calls. For this purpose, we use Kafka, which can disseminate jobs among various queues for tasks such as the view counter, notification system, analytics, and highlight topics to the user. Each of these jobs is executed through cron jobs.

Technology usage

Programming languages

Just like we mentioned that YouTube chose Python for faster programming, we can apply the same logic to Quora. In fact, Quora uses the Python Paste web framework.

It is desirable to use a faster programming language like **C++** to develop the **feature extraction service**. For online recommendation services through a ML engine, feature extraction service should be quick, to enable the ML engine to accomplish accurate recommendations. Not only that, but reducing the latency burden on the ML engine allows it to provide a larger set of services. We can employ the **Thrift** service to support interoperability between programming languages within different components.

Long polling

Features like comments, upvotes, and downvotes require frequent page updates from the client side. **Polling**is a technique where the client (browser) frequently requests the server for new updates. The server may or may not have any updates but still responds to the client. Therefore, the server may get uselessly overburdened. To resolve this issue, Quora uses a technique called **long polling**, where if a client requests for an update, the server may not respond for as long as **60** seconds if there are no updates. However, if there is an update, the server will reply immediately and allow the client to make new requests.

Mutiget()

Lastly, Memcached can employ `multiget()` to obtain multiple keys from the cache shards to reduce the retrieval latency of multiple keys

> ***Note:*** *Quora has employed AWS to set up a good number of its infrastructure elements, including S3 and Redshift storage.*

Final Design

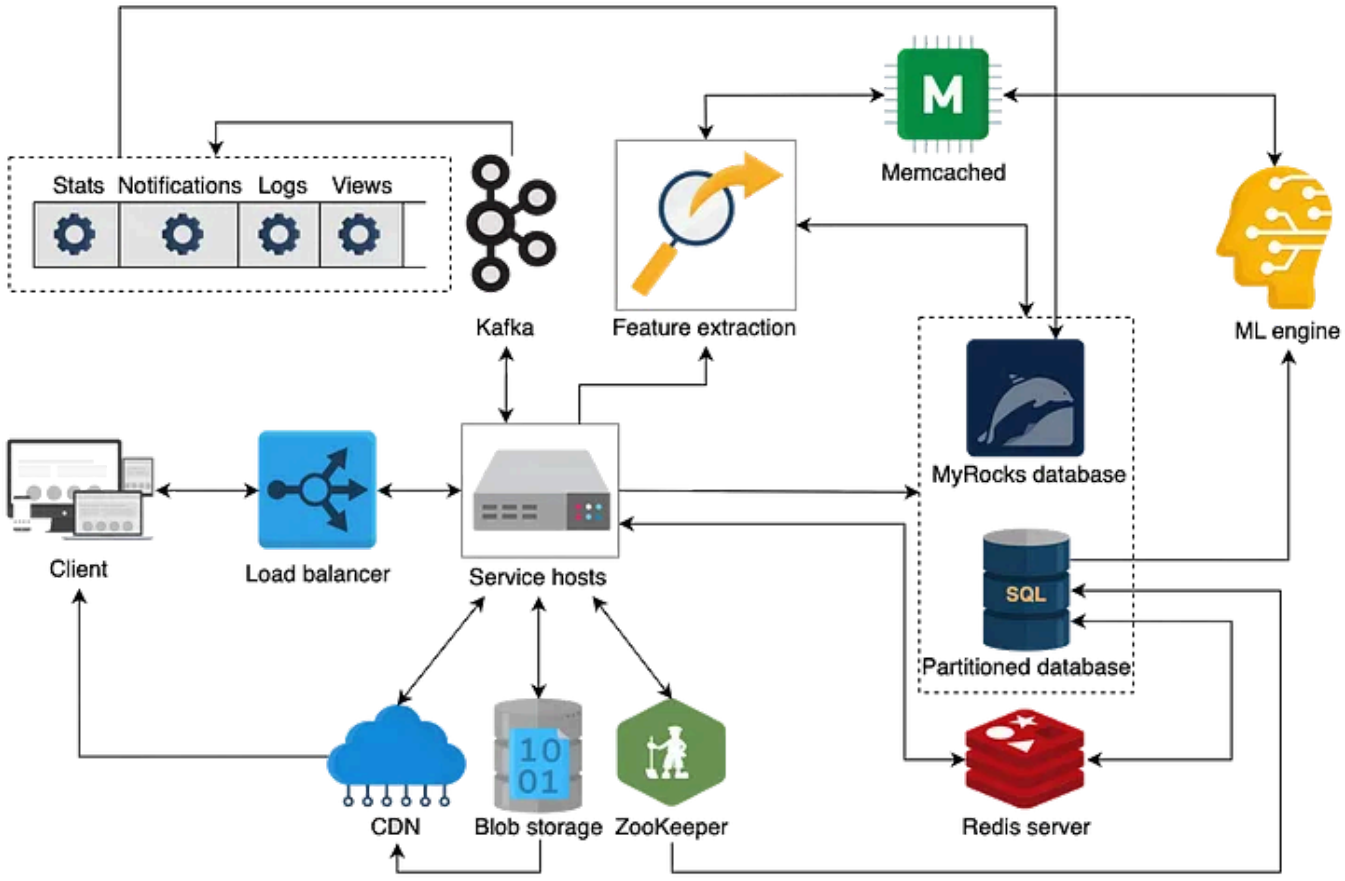


Fig 5.0: Detailed design of Quora

Evaluation

Scalability

Quora uses powerful machines because service hosts use an in-memory cache, some level of queueing, maintain manager, worker, and routing library. The horizontal scaling of these service hosts is convenient because they are homogeneous.

On the database end, our design shards the MySQL databases vertically, which avoids issues in scalability because of overloaded MySQL servers. To reduce

complex `join` queries, tables anticipating `join` operations are placed in the same shard or partition

Consistency

Due to the variety of functionalities offered by Quora, different consistency schemes may be selected for different types of data. For example, certain critical data like questions and answers should be stored synchronously. In this case, performance can take a hit because users don't expect instantaneous responses to their questions. It means that a user may get a reply in five minutes, one hour, one day, or no response at all, depending on the user's question and the availability of would-be respondents.

Other data like view counts may not necessarily be stored synchronously because it is not a goal of the Quora service to ensure that all users see the same number of views as soon as the question is posted. For such cases, **eventual consistency** is favored for improved performance.

Availability

Some of the main ideas to improve availability include isolation between different components, keeping redundant instances, using CDN, using configuration services like ZooKeeper, and load balancers to hide failures from users.

Performance

This design has a strong performance because we have employed the right technology for the right feature. For example, we have used several datastores for different reasons. On top of that, we used different distributed caches depending upon the use case and access frequency. Also, we employed Kafka to queue similar tasks and assign them to cron jobs that otherwise take a long time if executed via API calls.

Quora | System Design Interview | Software Architect | Software Architecture | Distributed System Design

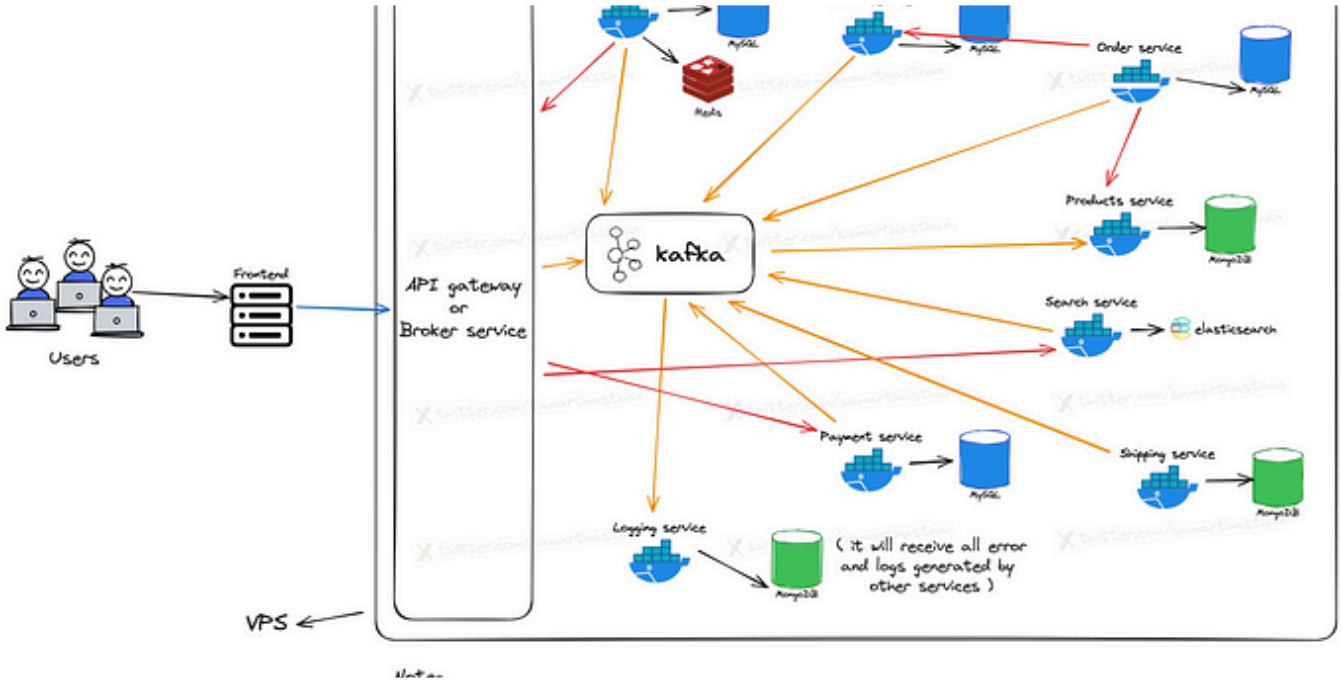
Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium



Samarth Asthan

Building a Scalable E-commerce Empire: A Micro-services System Design Approach

Hey everyone! I'm Samarth Asthan, a 3rd-year computer science student, and I'm new to the world of system design. But, I'm excited to share...

3 min read · Dec 11, 2023

8